Hans Journal of Data Mining 数据挖掘, 2021, 11(4), 226-240
Published Online October 2021 in Hans. http://www.hanspub.org/journal/hjdm
https://doi.org/10.12677/hjdm.2021.114021

Web数据集成中缺失数据处理方法研究

袁辉英，李　贵，李征宇，韩子扬，曹科研
沈阳建筑大学，信息与控制工程学院，辽宁 沈阳

收稿日期：2021年9月20日；录用日期：2021年10月22日；发布日期：2021年10月29日

摘　要

数据预处理是web数据集成中的一个重要步骤，修复缺失数据是数据预处理的重要组成部分。在web数据集成中修复缺失数据的关键问题是缺失点没有可直接提供参考的观察值，这导致用户不能使用估算和推理的方法，只能依靠有经验的用户或领域专家通过制定规则才能填充数据。然而，对于具有成千上万个缺失点的大型数据库，由用户理解数据并制定有效的填充规则是不可行的。因为在修复缺失数据时，用户需要了解哪些候选子集对缺失点填充概率和覆盖程度最大。然而，给用户推荐填充概率和覆盖程度最大的候选子集计算量非常大。为了解决这个问题，本文提出了一种基于信息熵的生成候选子集算法，通过用户对初始候选子集的编辑，使用该算法计算出缺失点填充概率和覆盖程度最大的候选子集。通过用户选择的候选子集并依据数据集中一对多的关联关系生成并推荐缺失点覆盖程度更高的规则，并将用户选择的规则通过数据集中一对多的关联关系泛化至更多的缺失点中。经过原型系统实现结果表明，用该方法修复的数据具有较高的精度，同时，实验表明普通用户在短时间内便可修复大量缺失数据，有效地提高了数据修复的效果。

关键词

数据预处理，Web数据集成，候选子集，缺失点，信息熵

Research on Missing Data Processing Methods in Web Data Integration

Huiying Yuan, Gui Li, Zhengyu Li, Ziyang Han, Keyan Cao
School of Information & Control Engineering, Shenyang jianzhu University, Shenyang Liaoning

Received: Sep. 20th, 2021; accepted: Oct. 22nd, 2021; published: Oct. 29th, 2021

Abstract

Data preprocessing is an important step in data integration, and repairing missing data is an im-

文章引用: 袁辉英, 李贵, 李征宇, 韩子扬, 曹科研. Web 数据集成中缺失数据处理方法研究[J]. 数据挖掘, 2021, 11(4): 226-240. DOI: 10.12677/hjdm.2021.114021

portant part of data preprocessing. The key problem in repairing missing data in data integration is that the missing points have no observations that can directly provide reference. This makes users unable to use estimation and reasoning methods, and can only fill in data by relying on experienced users or domain experts to formulate rules. However, for a large database with thousands of missing points, it is not feasible for users to understand the data and formulate effective filling rules. Because when repairing missing data, users need to know which candidate subsets have the greatest filling probability and coverage of missing points. However, it is very computationally intensive to recommend the candidate subset with the largest filling probability and coverage to the user. In order to solve this problem, this paper proposes an algorithm for generating candidate subsets based on information entropy. Through the user's editing of the initial candidate subsets, the algorithm is used to calculate the missing point filling probability and the candidate subset with the largest coverage. The rules with higher coverage of missing points are generated and recommended based on the candidate subset selected by users and the one-to-many association relation in the data set, and the rules selected by users are generalized to more missing points through the one-to-many association relation in the data set. The results of the prototype system implementation show that the data repaired by this method has high accuracy. At the same time, experiments show that ordinary users can repair a large number of missing data in a short time, which effectively improves the effect of data repair.

Keywords

Data Preprocessing, Web Data Integration, Candidate Subsets, Missing Points, Entropy

1. 引言

在 web 数据集成中需要进行分析处理的数据集通常是不完整的，其原因大致可分为两类：a) 由于程序缺陷或人为失误所造成的随机数据缺失；b) 数据已知但未及时录入数据库的数据。处理 a 类缺失数据的传统方法是通过插补或机器学习，由于插补和机器学习需要基于观察值，当某类实例数据已知但未录入数据库时，观察到的数据中将不包含这类实例的数据，所以不能使用推测的方法来修复 b 类缺失数据。同时，领域专家在编辑某类缺失点时，选择的候选子集往往能适用于多个缺失点，但在以前的系统中并没有关于显示候选子集缺失点覆盖范围的研究，用户只能多次填写同一候选子集，这极大地影响了修复数据的效率。因此，本文提出了一种基于信息熵的生成候选子集算法来筛选具有最大填充概率和覆盖程度的候选子集，并依据数据集中一对多的关联关系生成规则、泛化规则来帮助用户迭代地填充缺失数据。

1.1. 主要实例

本文引用了实际项目中的城市房地产大数据，该数据是利用网络爬虫从各城市房地产交易备案信息网站所爬取，并经过数据集成处理的真实数据。该实例展示了如何利用数据中一对多关联关系生成规则、泛化规则并应用数据库的过程。在用户填充缺失数据过程中，利用楼盘表、楼栋表、户表、许可表、栋许可关联表和开发商楼盘关联表等表中的一对多关联关系，楼盘表、楼栋表和户表可以使用嵌套方式引用，通过对用户编辑的数据使用基于信息熵的生成候选子集算法给用户推荐缺失点的候选子集。在本文实例中所用表相关属性如下所示：

楼盘表<楼盘名称(Name)、宗地编号(Dk-id)、建筑面积(area)、楼栋数(builds)、总套数(number)、车位数(P-count)>，简称“P 表”。

楼栋表<楼栋编号(D-id)、楼盘名称(Name)、楼栋地址(address)、楼层(floor)、总套数(number)>，简称“D 表”。

户表<楼栋编号(D-id)、楼盘名称(Name)、户编号(H-id)、建筑面积(area)、户型(type)>，简称“H表”。

许可表<楼盘名称(Name)、许可证号(L-id)、宗地编号(Dk-id)、楼栋数(builds)、总套数(number)>，简称“License表”。

栋许可关联表<楼盘名称(Name)、楼栋编号(D-id)、许可证号(D-L-id)>，简称“D_License 表”。

开发商楼盘关联表<楼盘名称(Name)、开发商名称(Developers)>，简称“P_Developers 表”。

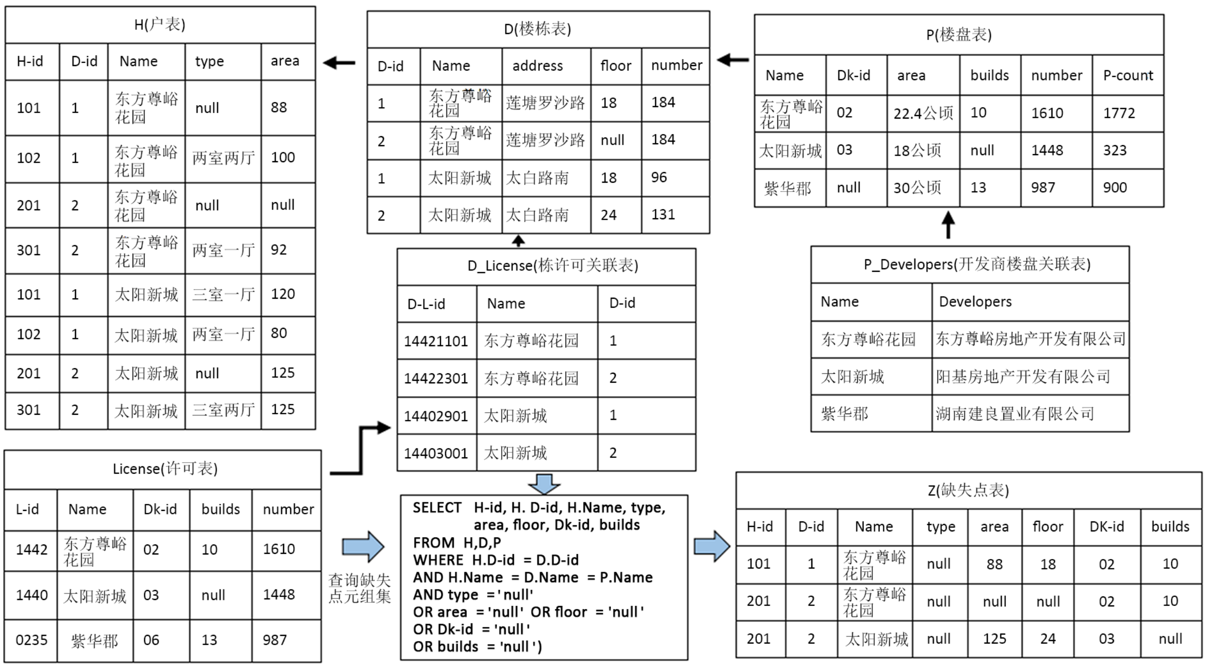

Figure 1. Urban real estate big data research example
图 1. 城市房地产大数据研究实例

如图 1 所示，在“户表中东方尊峪花园 1 栋 101 户中‘户型’的属性值缺失”，而数据中又没有可供估算和推理该元组的直接观察值。如果要通过辅助信息对这一缺失点进行填补，需要了解楼盘表、楼栋表和其他相关表属性与户表属性之间的关联关系。然而，在一对多关联关系中，通过各个表中的外键便可以创建一个非常宽的非规范化表，导致用户难以对缺失点的值进行推测，且由于事先并不了解数据，用户耗费大量精力在多个缺失点中填充同一候选子集。因此，为了提高修复缺失的效率。在填充数据时需要给用户推荐填充概率最大的候选子集，用户在选择每个候选子集时还需要了解候选子集对缺失点的覆盖程度。所以，本文需要研究的内容如下：

引导用户选择最有效的更新：在包含大量缺失数据的数据库中，如果用户不知道数据库中的信息情况，将难以编辑有效的更新规则来修复数据。例如：在城市房地产大数据的数据库中，用户填写户型属性的缺失值时指定了一个更新规则：当面积属性为“70 * 95 平”时，将户型的缺失值设置为“两室一厅”。如果在户型属性缺失的元组中面积属性也大部分缺失，则此更新的覆盖程度将非常小。因此，用户为了获得更有效的更新规则，需要使用其他属性，如：户编号、楼盘名称和楼栋编号属性。回到上面的例子，

东方尊峪花园 1 栋 101 户“户型”属性值缺失，可以设置为和东方尊峪花园 1 栋户编号为“_01”元组户型一致的更新规则：

UPDATE H SET type

WHERE type IN (SELECT type FROM H

WHERE D-id='1' AND H.Name='东方尊峪花园' AND H-id='_01');

根据更新规则结合户表信息可知，东方尊峪花园 1 栋 101 户户型属性值为“两室一厅”。然而，由于候选子集“两室一厅”可填充的缺失点并非只有东方尊峪花园 1 栋 101 户。因此，用户需要获得候选子集对缺失点覆盖程度查询。

利用一对多关联关系进行规则泛化：在修复数据的过程中，用户发现许多适用于子属性的规则也适用于其父级。因此，用户在选择规则时还应该向用户推荐适用于更大范围的更新规则，这可以利用数据库中一对多关联关系实现。例如，用户更新“东方尊峪 1 栋 101”的户型时，系统推荐在面积属性为“70 * 95 平”时，将户型设置为“两室一厅”的更新规则：

UPDATE H SET type='两室一厅'

WHERE D-id =1 AND Name='东方尊峪花园'

AND area BETWEEN 70 AND 95;

同时系统还推荐在“东方尊峪”楼盘中将面积属性为“70 * 95 平”时，户型设置为“两室一厅”的广义更新规则。

1.2. 本文主要工作

为了解决上述问题，本文提出了一种基于信息熵的生成候选子集算法，该算法用来实现筛选缺失点的候选子集，以 $p \times q$ 矩阵形式向用户显示了具有最大填充概率和覆盖程度的候选子集。用户在选择候选子集后，会根据生成规则算法向用户推荐应用于数据库的更新规则。如果用户在推荐的规则列表中看到合适的规则，可以立即将其应用于整个数据库。用户在编辑子集和选择规则之间来回切换，直到填充完所需的缺失数据。如果规则不再能够更新某类元组中的缺失值，则可以选择查看其他元组。通过实验验证该方法能有效地填补数据库中的缺失数据，同时减小了人力输入的消耗。本文的贡献主要包括：

1) 提出了一种基于信息熵的生成候选子集算法用于筛选缺失点填充概率和覆盖程度最大的候选子集。

2) 通过用户选择的候选子集，利用生成规则算法生成并推荐具有缺失点覆盖程度的候选规则，之后将用户选择的规则应用于数据库。

3) 通过数据库中一对多关联关系将用户选择应用的规则泛化至更多的缺失点中。

4) 通过一个原型系统向用户展示进行数据修复的过程并进行实验。实验结果表明，基于信息熵的生成候选子集算法显著提高了填充数据库中缺失数据的整体效率，平均迭代次数在 8 次左右便能填充大约 80%的缺失点，并保持了较高的精度。

2. 相关研究

Falcon [1]是一个减少用户工作量的交互式系统，该系统在更新单个缺失点前会向用户询问多个问题，从而通过放宽对 WHERE 子句的限制条件来概括最通用的更新规则，但无法给用户推荐具有最大填充概率和覆盖程度的候选子集。其推荐的规则主要用于错误检测和填充缺失数据，无法引导用户选择缺失点覆盖程度高的更新规则。同时，Falcon 测试数据集中的最大属性数是 15 个属性，无法浏览数据集中的长表和宽表。SampleClean [2]是一个基于查询的语义来处理脏数据的数据清洗系统，主要处理重复数据和

错误，而不是缺失数据。该系统通过归纳总结的方法来处理数据中的错误，并依据错误样本来查找数据库中的错误。HoloClean [3]是一个基于规则来识别和纠正错误的系统，主要结合了统计学习和概率推理的方法，通过对已知信息使用违反函数依赖关系(CFD)和完整性约束来修复数据。Potter'sWheel [4]是一个使用约束条件转换冲突的系统，Polaris [5]是一个基于用户编辑来推测规则的系统，这两种系统都没有向用户显示数据库样本，也没有使用查看各种元组来推测规则。Trifacta [6]系统提出了基于用户编辑的一般转换规则，但该系统侧重于属性提取，而不是填充缺失数据。ActiveClean [7]系统结合了机器学习和人工输入规则的方法，通过建立基础模型来推荐规则，并选择用用户检验数据的方法来提高数据模型的准确性，然而，当系统缺失特定的数据样本时，推荐给用户的规则并不适用。表 1 总结了相关系统之间的比较情况。

Table 1. Comparison of related systems
表 1. 相关系统的比较

系统	直接编辑	系统引导	生成规则	局限性
Falcon	√	×	√	不引导用户进行编辑
SampleClean	×	×	√	仅通过查询解决聚合的重复数据和错误，不能修复缺失数据
HoloClean	×	×	×	缺少有价值的数据信息导致精度或召回率低
Trifacta Poter's Wheel Polaris	√	×	√	尽管这些系统概括了基于编辑生成转换规则的方法， 但并未引导用户进行数据交互
ActiveClean	×	√	√	根据基础模型推荐规则，当缺失特定数据样本时，该规则不适用

3. 形式化描述及算法

设 D 为规范化的数据库模式，其中包含多个关系。每个关系 $X \in D$ 定义在一组属性 attr(X)上，属性 $A \in \mathrm{attr}(X)$ 的值域用 dom(A)表示。

3.1. 形式化描述

3.1.1. 更新规则

本文中更新规则的语句用 SQL 语言表示，如下所示：

UPDATE X SET A = a WHERE A = null AND B IN S

其中 $X \in D$， $A, B \in \mathrm{attr}(X)$。 $S \subset \mathrm{dom}(B)$ 可以是一组常量，也可以是在数据库 D 中提取的能与 X 关系中的 SELECT 查询。WHERE 子句可以进行扩展，从而约束 X 中的多个属性。

3.1.2. 规则泛化

用户在给定单个关系的情况下更新缺失值时，推荐的规则一般只考虑该关系的属性。而给定一个规范化的数据库时，可以利用一对多关联关系向用户推荐覆盖程度更大的规则。

给定一个关系 $R \subseteq X \times Y$，其中 R 关系是对 X 关系和 Y 关系进行多对多连接，$x\_id, y\_id \in \mathrm{attr}(R)$ 分别是 X 和 Y 的外键。假设 $r \in \mathrm{attr}(R)$ 中包含缺失值，将 R 关系显示给只使用 x_id 属性或 y_id 属性的用户无助于修复数据，因为该用户并不了解 R 中其他属性之间的关系。因此，应该从引用的关系中向用户显示一个有意义的属性，该属性可以由用户手动指定，也可以是最接近该关系的属性。在规范化数据库模式中的层次结构通常以一对多关联关系的形式表示，这样可以有利于用户理解属性之间的关系，也可以使用这些关联关系来减少数据修复过程中的人力消耗。

在一对多关联关系中，可以将多方的关系视为一方关系的子级。因此，存在一个层次结构，其中单

DOI: 10.12677/hjdm.2021.114021

侧关系是多侧的父级，并且适用于子级属性的规则可以泛化至父级属性中。例如，东方尊峪花园中户面积为“70 * 95 平”所对应的户型为“两室一厅”，该更新规则不仅适用于东方尊峪花园，还适用于该城市某区中所有的楼盘。定义如下：

定义 1 有关系 R、A 和 B，其中 $R \subseteq A \times B$。如果 R 是 A，B 之间的多对一连接，则 B 是 A 的父级。表示为： $B = \mathrm{Parent}(A)$。如果 $\forall a \in \mathrm{dom}(A)$， $b_2, b_2 \in \mathrm{dom}(B)$，则有：

$$(a, b_1) \in R \wedge (a, b_2) \in R \to b_1 = b_2$$

该定义表示适用于子级属性的规则可以泛化应用至父级属性中。

定义 2 有关系 $A, B, C \in D$，如果 B 是 A 的父级，C 是 B 的父级，那么对适用于 A 关系中属性规则可以推广到 B 关系的属性中，适用于 B 关系中属性规则可以推广到 C 关系的属性中，所以，适用于 A 关系中属性规则可以推广到 C 关系的属性中。用“ $\Rightarrow$ ”来表示“可以推广”，则有：

$$\mathrm{Parent}(A) = B \rightleftharpoons \mathrm{Rules}(\mathrm{attr}(A)) \Rightarrow \mathrm{Rules}(\mathrm{attr}(B))$$

$$\text{传递泛化：} \mathrm{Parent}(A) = B, \mathrm{Parent}(B) = C$$

$$\rightleftharpoons \mathrm{Rules}(\mathrm{attr}(A)) \Rightarrow \mathrm{Rules}(\mathrm{attr}(B))$$

$$\wedge \mathrm{Rules}(\mathrm{attr}(B)) \Rightarrow \mathrm{Rules}(\mathrm{attr}(C))$$

$$\rightleftharpoons \mathrm{Rules}(\mathrm{attr}(A)) \Rightarrow \mathrm{Rules}(\mathrm{attr}(C))$$

称父级的父级为祖父级，在图 1 所示的例子中，H (户表)中 type 和 area 属性的规则可以推广到该户所在的 D (楼栋表)，也可以通过楼栋表推广到该栋所在的 P (楼盘表)。如果用户在 Z (缺失点表)中东方尊峪花园 1 栋 101 户 type 属性中选择候选子集为“两室一厅”，则对应 H (户表)中东方尊峪花园 1 栋 101 户 type 属性数据，将更新为“两室一厅”，同时给用户显示的对应规则。内容如下：

```
UPDATE H SET type='两室一厅'
WHERE D-id IN (SELECT D-id FROM D
WHERE D-id =1 AND Name='东方尊峪花园')
AND area='88';
```

翻译成“东方尊峪花园 1 栋面积为 88 平的房子户型是‘两室一厅’”；

```
UPDATE H SET type='两室一厅'
WHERE D-id IN (SELECT D-id FROM D JOIN P
WHERE Name='东方尊峪花园')
AND area='88';
```

翻译成“东方尊峪花园楼盘里面积为 88 平的房子户型是‘两室一厅’”；

```
UPDATE H SET area='88'
WHERE D-id IN (SELECT D-id FROM D
WHERE D-id =1 AND Name='东方尊峪花园')
AND type='两室一厅';
```

翻译成“东方尊峪花园 1 栋户型是‘两室一厅’的房子面积为 88 平”；

这些规则中的 WHERE 子句作为一对多关联关系的连接键，且独立于此关系中的其他元组。因此，称该规则为独立规则。在部分数据更新完之后，用户可以通过数据之间的一对多关联关系查看剩余的 Z (缺失点表)。规则如下：

```
SELECT H-id, H.D-id, H.Name, type, area, floor, Dk-id, builds
FROM H,D,P
WHERE H.D-id=D.D-id AND H.Name=D.Name=P.Name AND type='null'
OR area='null' OR floor='null' OR Dk-id='null' OR builds='null';
```

以 Name 和 D-id 为核心属性，并为指定的存在缺失点的其他属性创建列，并按 H-id 属性分组，可得到缺失点表 Z。用户可以针对不同属性的缺失点填充不同的值，下面的规则也可以添加到候选的规则列表中：

```
UPDATE H SET area='92'
WHERE D-id='2' AND Name='东方尊峪花园'
AND H-id='_01' AND type='两室一厅';
```

翻译成“东方尊峪花园 2 栋”处于_01 位置的房子户型为“两室一厅”时，面积为 92 平；

这些规则可以利用数据库语言进行查询、配置和更新。WHERE 子句依赖于正在更新的关系中的其他元组，如东方尊峪花园 2 栋 201 户，面积属性填补规则依赖于户型属性，需要在用户填补户型属性后才能应用。因此，把依赖于独立规则更新数据后才能应用的规则为依赖规则。依赖规则仅在相似度高于阈值的属性之间生成。属性相似度可以是用户定义的，例如共享父级的公共外键，也可以使用 CORDS [8] 算法计算。

需要注意的是，更新规则需要有一个 $t_i[m] = \text{null}$ 的附加子句，即规则从不更新已有或已用先前规则填充的数据。此外，在用户选择规则之前，规则不会应用。用户编辑的数据可能与源数据冲突，但这并不意味着规则或源数据不正确。例如，用户可以将东方尊峪 1 栋 501 户面积为 88 平的户型编辑为“一室一厅”，因为该户主在装修时将两个房间合并为一个房间。此外，用户可以先应用特定的狭义规则，然后再进行泛化。这类规则是针对特殊情况所制定的，因此不应覆盖先前的规则。

3.1.3. 迭代

在前面已经介绍了规则的形式化描述以及如何进行规则泛化。当用户在规范化的数据库中没有发现足够的参考信息来编辑数据时，需要通过非规范化的形式来获得信息。然而，数据集中元组数量超过一万条便使得计算代价非常大。因此，用户需要从非规范化的形式中获得侯选子集，同时用户还尽可能多地了解候选子集覆盖的缺失点数量。假设一个关系只有一个属性存在缺失值，给定一个存在缺失数据的关系 D，设 U 为 D 中包含缺失点的元组集合，S 为关系 D 中包含的候选子集集合。每个候选子集 $s_i \in S$ 可填充的缺失值元组为 $s_i \cap U$，在对该元组进行编辑时，会生成一组规则 $\text{rules}(s_i)$。如果规则 $r \in \text{rules}(s_i)$ 被用户接受，则将候选子集 s_i 填充在 $s_i \cap U$ 中，称从给用户显示候选子集到用户接受并应用规则的过程称为一次迭代。正式定义如下：

数据迭代修复问题：给定需要进行数据修复的关系 D，设 U 为 D 中包含缺失点的元组集合，数据迭代修复问题表示为使通过选择候选子集 $s_i \in S \subset D$ 来使填充 U 中数据所需要迭代次数最少或使每次迭代中缺失点覆盖程度 $s_i \cap U$ 最大。其迭代流程如图 2 所示。

用户编辑缺失点数据后，会根据基于信息熵的生成候选子集算法筛选出具有最大填充概率和覆盖程度的候选子集推荐给用户，并根据生成规则算法生成相应的规则供用户选择。当用户选择符合的候选子集和规则，并应用规范化的数据库后，完成一次迭代。图中阈值为缺失数据比例，可由用户定义。

3.2. 相关算法

在 3.1 中已经介绍了更新规则的表达形式、规则泛化的概念、修复数据的迭代过程。在本节中主要研究了与修复缺失数据相关的算法，包括基于信息熵的生成候选子集算法和生成规则算法。

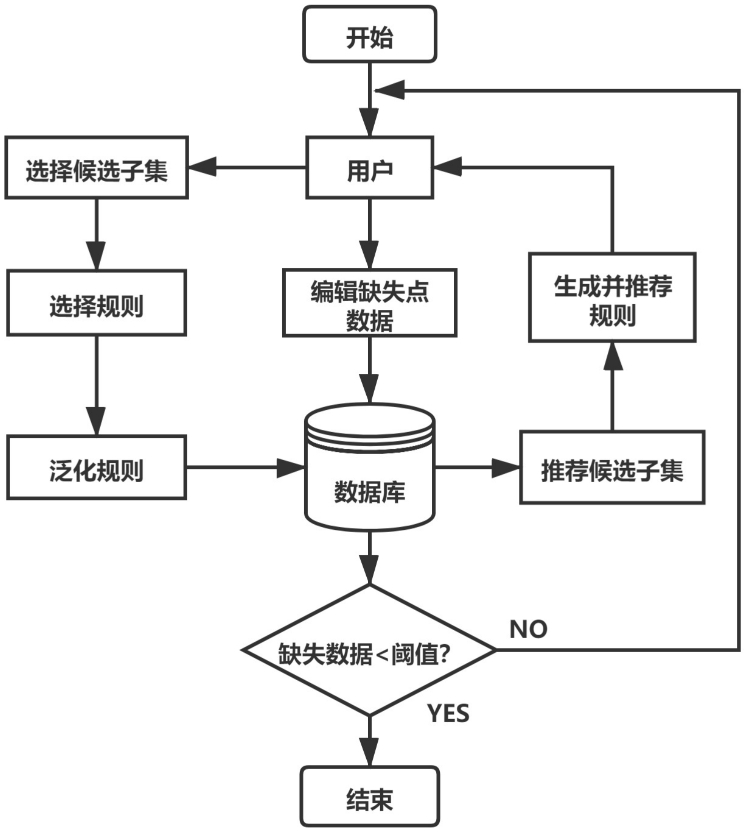

Figure 2. Iterative process
图 2. 迭代流程

3.2.1. 基于信息熵的生成候选子集算法

在填充缺失数据的过程中，领域专家发现存在多个缺失点填充值相同的情况，由于普通用户事先并不了解这一情况，只能依次对缺失点进行编辑修复，这不仅降低了修复数据的效率，还损耗了人力资源。因此，本文提出基于信息熵的生成候选子集算法，希望筛选出的候选子集可以使用户更新缺失点的概率及覆盖程度最大化。

用户选择候选子集更新缺失点的概率取决于数据库中是否具有足够的信息来给缺失点提供参考，且经过用户编辑填充后的缺失点对未填充的缺失点具有一定参考价值。给定一个非规范化的数据库 $Z = A\times B$，A 表示所有行，B 表示所有列，其中 $a\subset A$， $b\subset B$ 表示用户填充缺失点时具有参考价值的行和列，用户通过参考 $a\times b$ 中的信息使缺失点被更新的概率及其覆盖程度最大化。设 M 表示 Z 中缺失点集，N 表示 Z 中已有的子集集，包括填充后的缺失点子集，$m_{i,j}$ 表示 M 中单个缺失点，n_k 表示 N 中的单个子集，矩阵 $\mathrm{sim}\left[b_j\right]\left[b_k\right]$ 表示 j 列和 k 列之间的相似度，$\mathrm{Coverage}\left[b_j\right]\left[b_k\right]$ 表示 k 列子集存在 j 列子集缺失的行数，本文通过列与列之间相似度和用户填补缺失点可提供参考的行数的乘积来量化信息的参考价值，计算公式如下：

$$\mathrm{Scores} = \sum_{m_{i,j}\in\{z\cap M\}} \sum_{n_k\in\{a_i\cap N\}} \mathrm{sim}\left[b_j\right]\left[b_k\right]\cdot \mathrm{Coverage}\left[b_j\right]\left[b_k\right] \tag{1}$$

其中：缺失点集 M 和已有的子集集 N 均为非空集和，公式 1 表示所有缺失点被用户选择并填充的得分情况，k 列子集存在 j 列子集缺失的行数与 j 列和 k 列相似度成正比，得分越高，说明在这两列之间推荐的规则，将使候选子集对缺失点的填充概率和覆盖程度最大化。其中列相似度 $\mathrm{sim}\left[b_j\right]\left[b_k\right]$ 的计算方法可以

由用户定义，也可以使用 CORDS 算法[8]计算。

然而，仅凭列相似性和对缺失点的覆盖程度大小还不能够筛选出合格的候选子集。因为独立规则中特征属性不一定相关，因此，在筛选候选子集时，还需要衡量缺失点所在的行和列中的值在 Z 中其他行和列的值出现的概率，从而为用户提供参考依据。称某种特定信息的出现概率为信息熵，其计算如下：

$$H(X) = -\sum_{x \in X} P(x) \log P(x) \tag{2}$$

其中 X 表示为某一行或列中已有子集集，$P(x)$ 为每个子集在子集集 X 中出现的概率。信息熵值越高表示可以提供给用户的参考信息越多。因此，本文希望筛选出的候选子集不仅具有最大的填充概率和覆盖程度，还能具有较高的信息熵。然而，当用户填充需要大量的参考信息才能做决策的缺失点时，希望提供选择的候选子集越多。考虑到这一点，本文借鉴模拟退火技术的思想[9]，随着迭代次数的增加熵的权重，对公式 1 进行了优化，优化函数公式如下：

$$\text{Scores} = \sum_{m_{i,j} \in \{z \cap M\}} \sum_{n_k \in \{a_i \cap N\}} \text{sim}\left[b_j\right]\left[b_k\right] \cdot \text{Coverage}\left[b_j\right]\left[b_k\right] + H(z) \cdot \text{temp} \tag{3}$$

其中 temp 表示数据修复的迭代次数，由于针对具有列相似度的不同的行进行优化，使得筛选出具有上述条件的候选子集难度较大。为此，本文提出了基于信息熵的生成候选子集算法，如算法 1 所示：

算法 1 基于信息熵的生成候选子集算法

iterations:迭代次数

vis_a[i]:选择第 i 行迭代

entropy_a[i]:第 i 行的信息熵

Coverage[i][j]:j 列的值缺失但对应 i 列值存在的数量

$\text{entropy}_b^{\text{rows}}[i]$:所有行中 i 列值的信息熵

I[a][i]:如果 a 行 b 列中值存在，指标函数为 1，否则为 0

1:生成_候选子集(c)

2: A←c.rows

3: B←c.columns

4: temp=iterations/100

5: for i=1 to A.length

6:Scores_a[i] = iterations/vis_a[i] + entropy_a[i] + missing_a[i]

7: rows←scores_a 排列前 p 名的候选子集

8: for i=1 to B.length

9: $\text{Scores}[i] = \text{entropy}_b^{\text{rows}}[i] \cdot \text{temp} + \text{sim}[b][i] \cdot \text{Coverage}[b][i] \cdot \sum_{a \in \text{rows}} \text{I}[a][i]$

10: columns←Scores_b 排列前 q 名的候选子集

11: c←rows∩columns

12：rerurn c

本文分为两个阶段对所有候选子集进行评分，首先将行和列中的已有子集赋值给 A、B 所在的序列，如步骤 2，3 所示。步骤 5 至步骤 7 为第一阶段，然后将候选子集所在行的信息熵、缺失点数量和迭代次数作为行评分标准来筛选候选子集所在的行。步骤 8 至步骤 10 为第二阶段，利用列相似度和覆盖程度成正比计算缺失点被用户选择并填充的得分，并加入随着迭代次数的增加熵的权重的思想作为列评分标准，如公式 3 所示。步骤 11 筛选出高行评分和高列评分的所在的候选子集，供给用户选择。

3.2.2. 生成规则算法

给定数据库 $M = X \times Y$，用户将非规范化表中缺失点 $t_i[m]$的值填充为 n，表示为$t_i[m] = n$，其中$m \in \mathrm{attr}(M)$，表示 m 为数据库 M 中的某一属性。当外键 x_id 和 y_id 与非规范化表中的 $t_i[x_id]$和 $t_i[y_id]$具有共享父级或者传递关系时，那么用户在非规范化表中填充的每一列 $t_i[b]$对应于共享父级的每一列 b。表示为当$b = t_i[b]$时，$t_i[m] = n$。具体算法如下：

算法 2 生成规则算法

```
m:数据库 M 中的某一属性
n:填充值
i:元组索引
ti[x_id]:属性 x_id 中元组 i 对应的值
ti[x_id].S.id:与 ti[x_id]相关联的属性 S.id 的值
数据库 M 中 X 与 Y 多对多联接
生成_独立_规则(m,n,i)
    S←X
    while S≠∅ do
        T←Y
        while T≠∅ do
            rules.append(UPDATE SET m=n WHERE x_id
            in(SELECT id FROM X WHERE S.id=ti[x_id].S.id) AND
            y_id in (SELECT id FROM Y WHERE T.id=ti[y_id].T.id))
            T←T.parent
            Y←Y×T
        end while
        S←S.parent
        X←X×S
    end while
    return rules
生成_依赖_规则(m,n,i)
```

Continued

```
15:  S←X
16:  A←{ y∈Y if y.parents∩t_i[y_id].parents≠∅∧
         M[t_i[x_id]][y.id] ≠null}
17:  for y∈A
18:      while S≠∅ do
19:          rules.append(UPDATE SET m=n WHERE x_id
         in(SELECT id FROM X JOIN M WHERE S.id=t_i[x_id].S.id
         AND M.y_id=y.id AND m=M[t_i[x_id]][y.id].m) AND y_id
         =t_i[y_id])
20:          S←S.parent
21:          X←X×S
22:      end while
23:  end for
24:  return rules
```

算法 2 分为两个部分，其中步骤 1 至步骤 13 为生成独立规则部分。当属性 S.id 和 T.id 与非规范化表中属性 x_id 和属性 y_id 共享父级时，将用户在非规范化表属性 t_i[b]中填充的值 n 赋值给第 i 行对应属性 b 中。当属性 S.parent 和属性 T.parent 为 S 和 T 的父级时，可以将应用于属性 S.id，T.id 的规则泛化至 S.parent 和 T.parent 中。步骤 14 至 24 为生成依赖规则部分。当数据库 M 中属性 x_id 和属性 y_id 值非空，且属性 y 与非规范化表中属性 y_id 的父级都为非空集合时，才执行生成依赖规则。当属性 S.id 依赖于共享父级属性 x_id 的，且该属性已经依靠独立规则进行填充后，将用户在非规范化表属性 t_i[b]中填充的值 n 赋值给第 i 行对应属性 b 中，同时将应用于属性 S.id 的规则泛化 S.parent 中。

4. 原型系统及实验

4.1. 原型系统

在本节中主要展示了原型系统的工作框架，该系统是一个旨在通过筛选最大填充概率和覆盖程度的候选子集，利用数据集中一对多关联关系来给用户推荐适用的规则，从而尽可能减少填充缺失数据工作量的系统。

如图 3 所示，该界面分为三个部分，右边显示缺失数据子集的编辑接口。在每次迭代中，以 $p \times q$ 矩阵的形式向用户显示缺失点中的候选子集，根据用户选择的候选子集在左上窗口显示推荐适用于更多缺失点的规则。同时，用户可以实时查看存在缺失数据的百分比。左边为规则显示栏，显示根据候选子集选择情况生成的规则和根据数据集中一对多关联关系泛化的规则，同时显示更新规则将覆盖的缺失点数量，左下角统计了数据集中各类属性存在的缺失点情况，供用户参考。

4.1.1. 工作流程算法

前面 3.1.3 节中的迭代流程图已经总结了在填补缺失数据的具体流程，当数据库缺失点数量达到阈值(可由用户定义)时，通过非规范化的数据库将缺失点情况反馈给用户，在用户编辑数据后，将结合用户编辑的数据根据候选子集生成算法显示给用户其他缺失点候选子集，用户可以选择其中一个候选子集填充缺失点。在填充缺失点时，会根据规则生成算法给用户推荐规则，该规则分为独立规则和依赖规则。如果用户看到合适的规则，选择接受后，将该规则应用到数据库。同时，根据规则的应用情况继续向用户

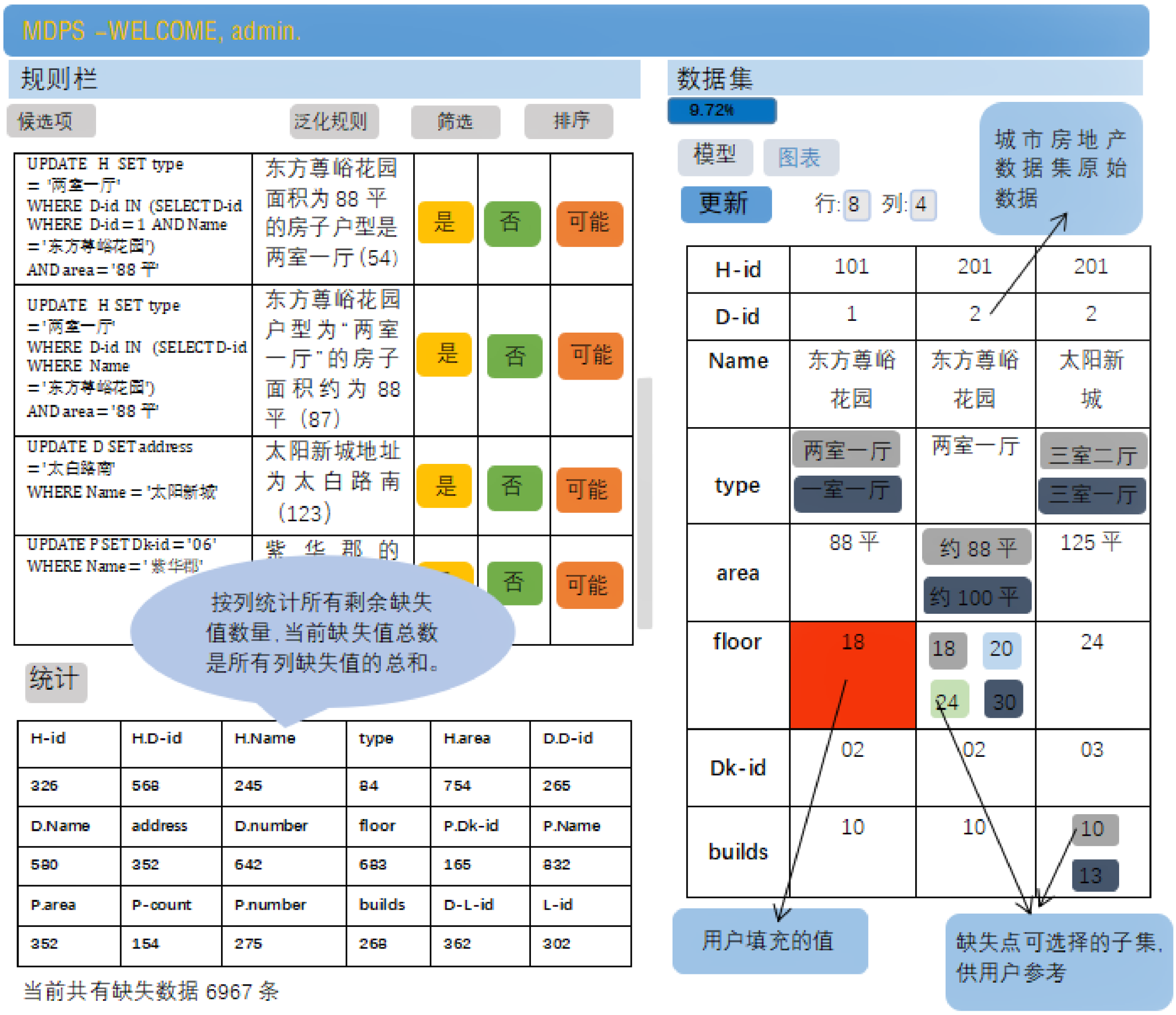

Figure 3. Prototype system interface
图 3. 原型系统界面

推荐规则泛化。如果用户觉得没有足够的信息做出决定，可以选择查看其他元组。用户可以从查询语言查看缺失点数量，然后用户继续填写缺失点并应用规则，直到剩余的缺失数据低于阈值为止，从而结束迭代。通过之前的分析本节给出了修复缺失数据的工作流程算法，如算法 3 所示：

算法 3　工作流程算法
c:数据库
e:阈值，可由用户定义
1：while 缺失数据比例> e do
2：　　while 用户编辑 do
3：　　　　c⇐t_i[m]=n
4：生成_候选子集(c)
5：规则=独立_规则(m,n,i)+依赖_规则(m,n,i)

Continued

```
6:          if 用户接受规则 then
7: c⟸规则
8:          end if
9:     end while
10:  end while
```

首先用户通过缺失数据比例阈值的定义开始修复数据的迭代过程，如步骤 1 所示。步骤 2 至步骤 5 为用户开始编辑缺失点数据，用户将 n 填充给缺失点 $t_i[m]$，并将填充结果反馈给数据库 c，根据生成_候选子集算法筛选具有最大填充概率和覆盖程度的候选子集，并生成对应的规则，包括独立规则和依赖规则。步骤 6 至 8 为用户接受并应用规则的过程。步骤 9 表示结束一次迭代过程，用户可以查看并编辑其他元组的缺失点。步骤 10 表示剩余的缺失点数量低于缺失数据比例 e，完成修复数据的任务。

4.1.2. 系统功能

用户可在原型系统界面上进行的交互包括：

更新缺失值：用户可以通过点击对应的候选子集来更新缺失值，表示为界面中的颜色按钮。更新时，将在左上栏中生成规则。

规则栏：规则栏是可滚动的，推荐的规则按缺失点覆盖程度从大到小排列，可以筛选仅显示独立或依赖的规则。

规则应用：用户看到合适的规则时，可以通过单击规则旁边的“是”或“可能”按钮接受规则。这两个按钮都能将规则应用于整个数据集，但是“可能”按钮对应的规则在存储时被称为“低置信度”规则。此反馈不用于该原型系统，仅当专家填补数据时用于比较分析结果。如果用户选择“否”按钮，则将不再生成该规则。

更新：用户可以通过单击界面顶部的“更新”按钮来查看不同的元组。

4.1.3. 优化

尽管该系统给用户呈现了一个非规范化的数据库形式，但整个数据库在实现过程中并不会被非规范化。下面介绍原型系统不同组件的详细信息。

选择候选子集：在选择候选子集过程中使用的每个组件，例如每行和每列缺失点、行熵、列熵、列相似度以及缺失点的覆盖程度，会提前计算并存储为单独的表。实际上，列熵只计算显示给用户的行，因此可以动态计算。

规则生成和应用：规则生成遵循一对多关联关系中父级的深度优先搜索。如果一个关系有较深的等级层次，那么深度可以限制在用户定义的 k 上，Falcon-dive 算法[1]可以用于限制推荐规则。在每次迭代中，只需要对推荐给用户的候选子集即执行此操作。因此，在每次迭代开始时，会为每个缺失点中的候选子集生成并缓存规则。

4.1.4. 时间复杂度分析

在每次迭代中，每行和每列缺失点、行熵、列熵、列相似度以及缺失点的覆盖程度等组件会提前计算，其中选择候选子集行的时间复杂度是 $o(m)$，列的时间复杂度是 $o(n)$。在非规范化表 $m \times n$ 中，设 x 为缺失点的百分比，则应用规则需要的时间复杂度为 $o(x\cdot m\cdot n)$，所以，总的时间复杂度为 $o(m+n+x\cdot m\cdot n)\equiv o(m\cdot n)$。

4.2. 实验

在本节中的实验结果主要为 2 个方面：(a)比较基于信息熵的生成候选子集算法与其他生成候选子集算法；(b)比较不同缺失比例下的数据修复情况；本文使用以下数据集：城市房地产数据集：这是本文示例中的数据集，包含某城市大约四十万条的房地产信息，包括楼盘表、楼栋表、户表、许可表、栋许可关联表和开发商楼盘关联表等各种类型数据。为了比较原型系统修复缺失数据的效果，该数据集人为删除了部分的缺失数据，比较不同缺失比例下的数据修复情况。

其它生成候选子集算法如下：

随机：在每次迭代中随机选择一个子集。

迭代随机：在每次迭代中将选择行和列的概率与已经迭代的次数成反比。

聚类：在每次迭代中根据行的相似性对行进行聚类，从聚类中提取与缺失值成比例的行，然后使用本文中的优化函数选择列。

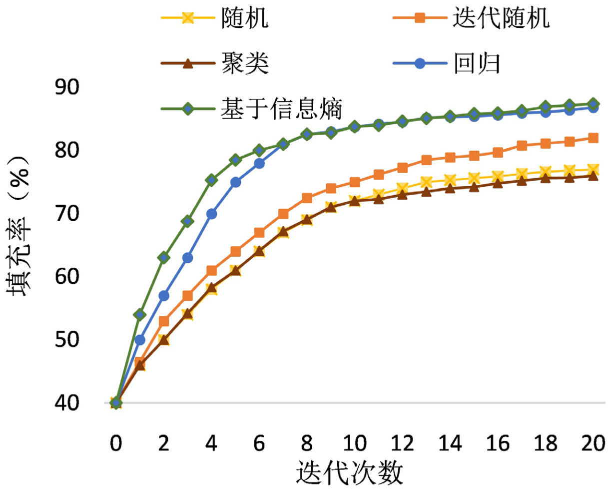

Figure 4. Comparison of 5 algorithms for generating candidate subsets
图 4. 5 种生成候选子集算法的比较情况

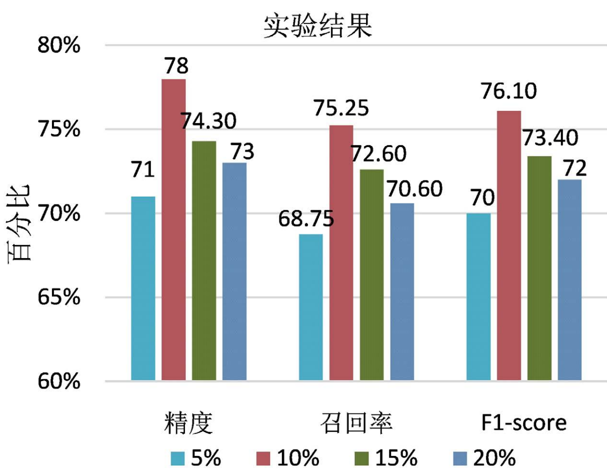

Figure 5. Comparison of accuracy, recall rate and F1-score at different missing ratios
图 5. 不同缺失比例下精度、召回率和 F1-score 的比较情况

回归：在每次迭代中选择缺失值最多的列进行优化。然后训练一个线性支持向量机(SVM) [10]，根据包含缺失值的列和模型的权重来选择得分靠前的$q-1$列，然后选择所选列中填充值最多的行。

实验中生成候选子集算法会生成多个候选子集，编辑其中一个缺失点会生成相应的规则，由用户选择是否应用该规则。对于每个算法，本文进行了 30 次测试，实验结果取其平均值。

图 4 显示 5 种生成候选子集算法在城市房地产数据集上的数据修复情况。从图中可以看到，基于信息熵的生成候选子集算法整体优于其它生成候选子集算法。在第 8 次迭代时，填充的数据达到所有缺失数据的 80%，数据修复效果显著提高。之后，对不同缺失数据比例下修复数据的精度(precision)、召回率(recall)和 F1-score 进行比较分析，实验结果显示，数据缺失比例在 10%左右，数据修复情况最好，如图 5 所示。

5. 总结

本文提出了一个基于信息熵的生成候选子集算法用于缺失点候选子集的筛选工作，并通过一个交互式的原型系统实现数据修复过程，该系统根据用户编辑的缺失点数据，以$p \times q$矩阵的形式向用户显示缺失点的候选子集，依据数据集中一对多关联关系生成并推荐缺失点覆盖程度更高的规则，用户在选择这些规则后，可以立即更新数据集。用户可以在编辑数据和应用规则之间来回切换，直到填充的缺失数据比例达到规定阈值为止。其中基于信息熵的生成候选子集算法不仅使用户更新数据的概率和覆盖程度最大化，而且提高了数据修复的效果，使得普通用户在较短的时间内便能修复大量缺失数据，减少了人力方面的消耗。

参考文献

[1] He, J., Veltri, E., Santoro, D., Li, G., Mecca, G., Papotti, P. and Tang, N. (2016) Interactive and Deterministic Data Cleaning. *Proceedings of the* 2016 *International Conference on Management of Data*, San Francisco, California, 26 June 2016-1 July 2016, 893-907. https://doi.org/10.1145/2882903.2915242

[2] Wang, J., Krishnan, S., Franklin, M.J., Goldberg, K., Kraska, T. and Milo, T. (2014) A Sample-and-Clean Framework for Fast and Accurate Query Processing on Dirty Data. *Proceedings of the* 2014 *ACM SIGMOD International Conference on Management of Data*, New York, NY, 22-27 June 2014, 469-480. https://doi.org/10.1145/2588555.2610505

[3] Rekatsinas, T., Chu, X., Ilyas, I.F. and Ré, C. (2017) Holoclean: Holistic Data Repairs with Probabilistic Inference. *Proceedings of the VLDB Endowment*, **10**, 1190-1201. https://doi.org/10.14778/3137628.3137631

[4] Raman, V. and Hellerstein, J. (1999) Potter's Wheel: An Interactive Framework for Data Cleaning. Technical Report, Working Paper. http://www.cs.berkeley.edu/rshankar/papers/pwheel.pdf

[5] Stolte, C., Tang, D. and Hanrahan, P. (2002) Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *IEEE Transactions on Visualization and Computer Graphics*, **8**, 52-65. https://doi.org/10.1109/2945.981851

[6] Kandel, S., Paepcke, A., Hellerstein, J. and Heer, J. (2011) Wrangler: Interactive Visual Specification of Data Transformation Scripts. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 3363-3372.

[7] Krishnan, S., Wang, J., Wu, E., Franklin, M.J. and Goldberg, K. (2016) Active Clean: Interactive Data Cleaning While Learning Convex Loss Models. *Proceedings of the VLDB Endowment*, **9**, 948-959. https://doi.org/10.14778/2994509.2994514

[8] Ilyas, I.F., Markl, V., Haas, P., Brown, P. and Aboulnaga, A. (2004) CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. *Proceedings of the* 2004 *ACM SIGMOD International Conference on Management of Data*, New York, NY, 13-18 June 2004, 647-658. https://doi.org/10.1145/1007568.1007641

[9] Aarts, E. and Korst, J. (1988) Simulated Annealing and Boltzmann Machines.

[10] Suykens, J.A. and Vandewalle, J. (1999) Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, **9**, 293-300. https://doi.org/10.1023/A:1018628609742